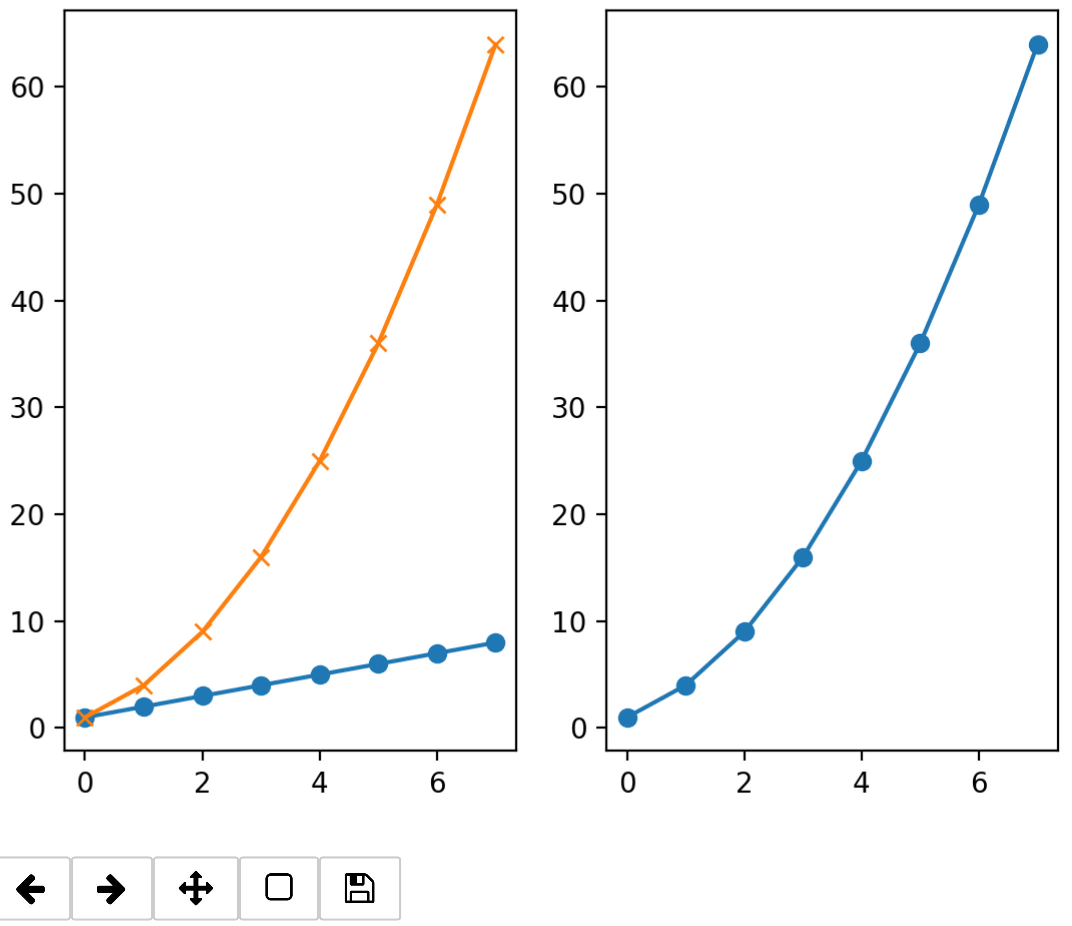
Subplots

In [1]:
```
%matplotlib notebook

import matplotlib.pyplot as plt
import numpy as np

plt.subplot?
```

In [2]:
```
plt.figure()
# subplot with 1 row, 2 columns, and current axis is 1st subplot axes
plt.subplot(1, 2, 1)

linear_data = np.array([1,2,3,4,5,6,7,8])

plt.plot(linear_data, '-o')
```

Figure 1



Out[2]: [<matplotlib.lines.Line2D at 0x7f63746486a0>]

In [3]:
```
exponential_data = linear_data**2

# subplot with 1 row, 2 columns, and current axis is 2nd subplot axes
plt.subplot(1, 2, 2)
plt.plot(exponential_data, '-o')
```

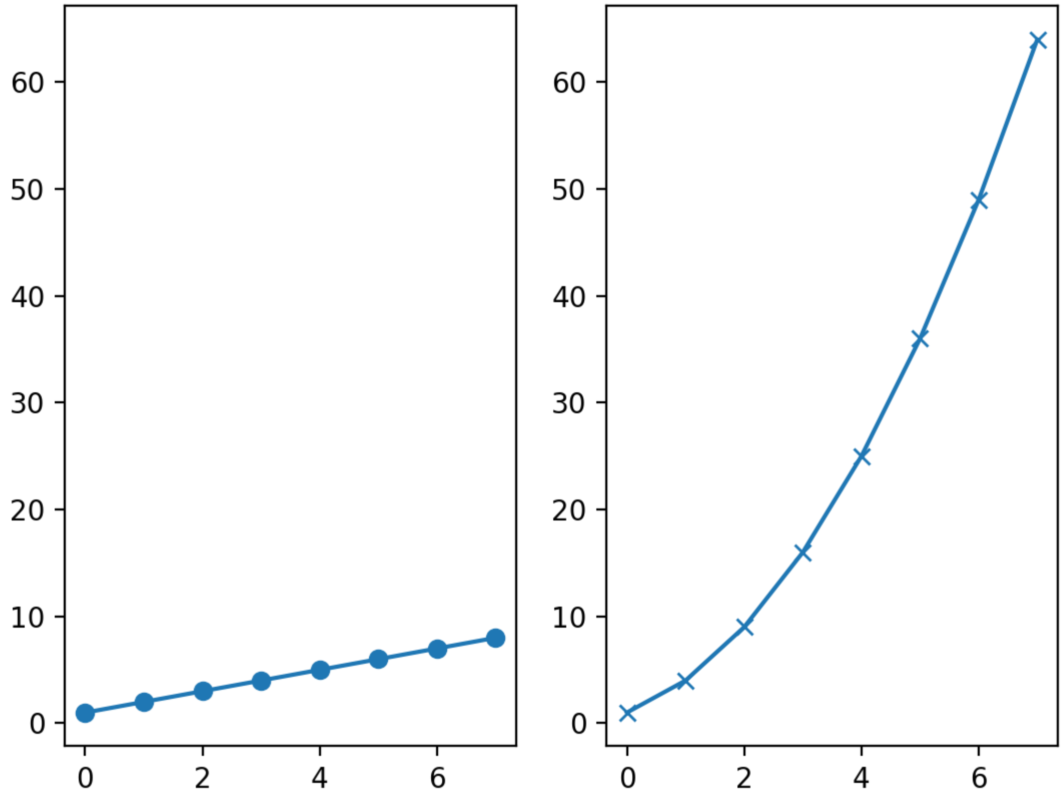
Out[3]:
```
[<matplotlib.lines.Line2D at 0x7f6343e902b0>]
```

In [4]:
```
# plot exponential data on 1st subplot axes
plt.subplot(1, 2, 1)
plt.plot(exponential_data, '-x')
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7f6343e90358>]
```

In [5]:
```
plt.figure()
ax1 = plt.subplot(1, 2, 1)
plt.plot(linear_data, '-o')
# pass sharey=ax1 to ensure the two subplots share the same y axis
ax2 = plt.subplot(1, 2, 2, sharey=ax1)
plt.plot(exponential_data, '-x')
```

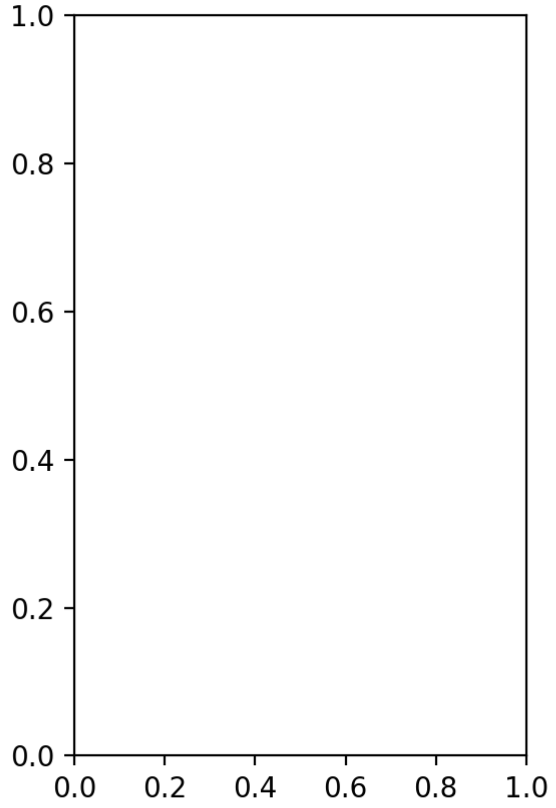
Figure 2



Out[5]: [<matplotlib.lines.Line2D at 0x7f6341af3fd0>]

In [6]:
```
plt.figure()
# the right hand side is equivalent shorthand syntax
plt.subplot(1,2,1) == plt.subplot(121)
```

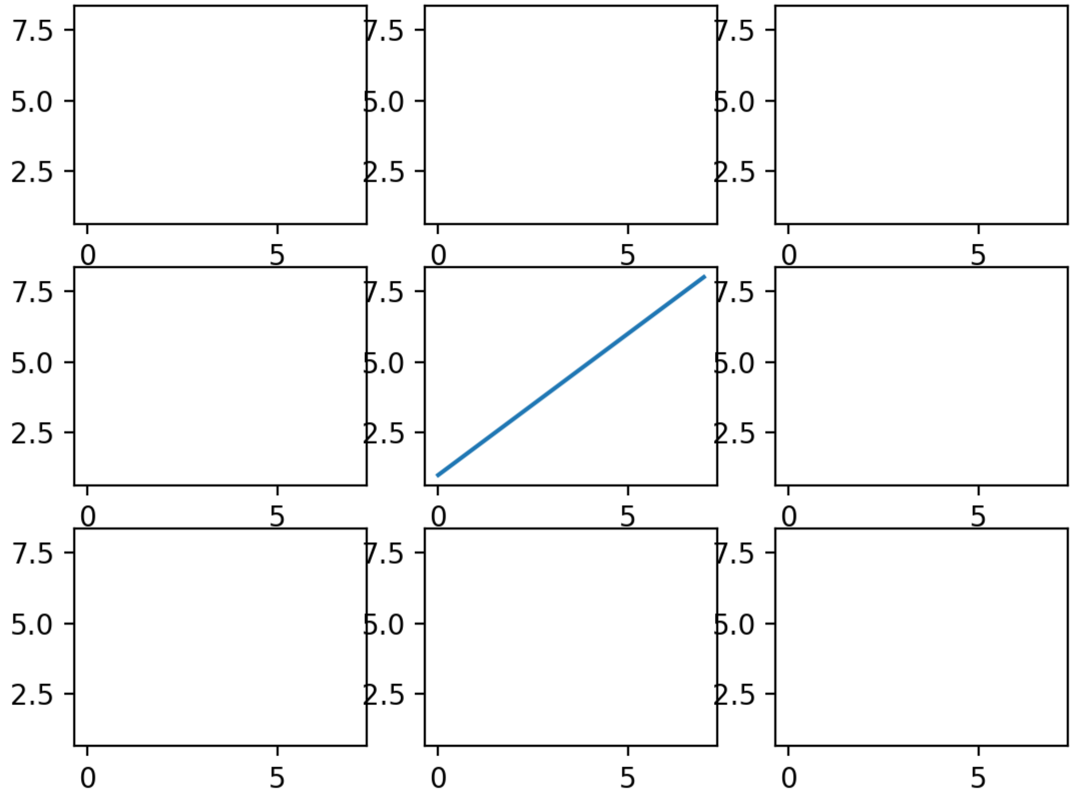
Figure 3



Out[6]: True

In [7]:
```
# create a 3x3 grid of subplots
fig, ((ax1,ax2,ax3), (ax4,ax5,ax6), (ax7,ax8,ax9)) = plt.subplots(3, 3, sharex=True, sharey=True)
# plot the linear_data on the 5th subplot axes
ax5.plot(linear_data, '-')
```

Figure 4



Out[7]:
```
[<matplotlib.lines.Line2D at 0x7f6341a0d4a8>]
```

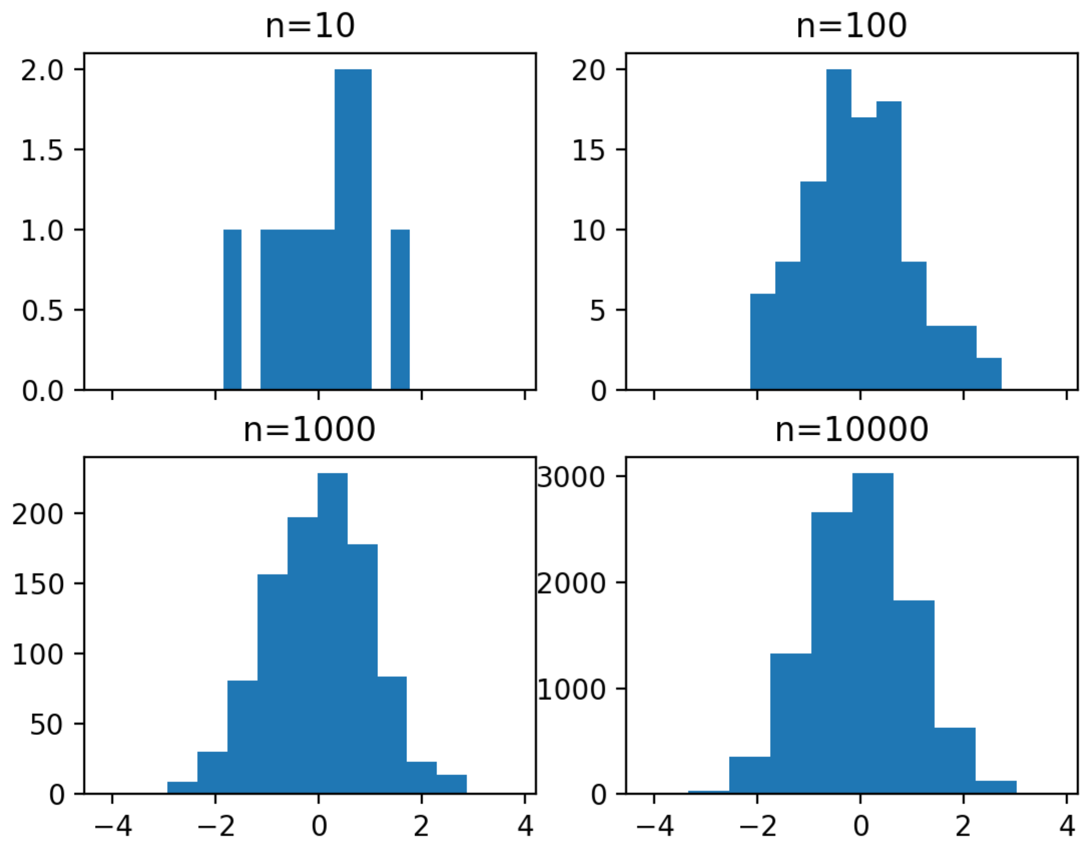
In [8]:
```
# set inside tick labels to visible
for ax in plt.gcf().get_axes():
    for label in ax.get_xticklabels() + ax.get_yticklabels():
        label.set_visible(True)
```

In [9]:
```
# necessary on some systems to update the plot
plt.gcf().canvas.draw()
```

Histograms

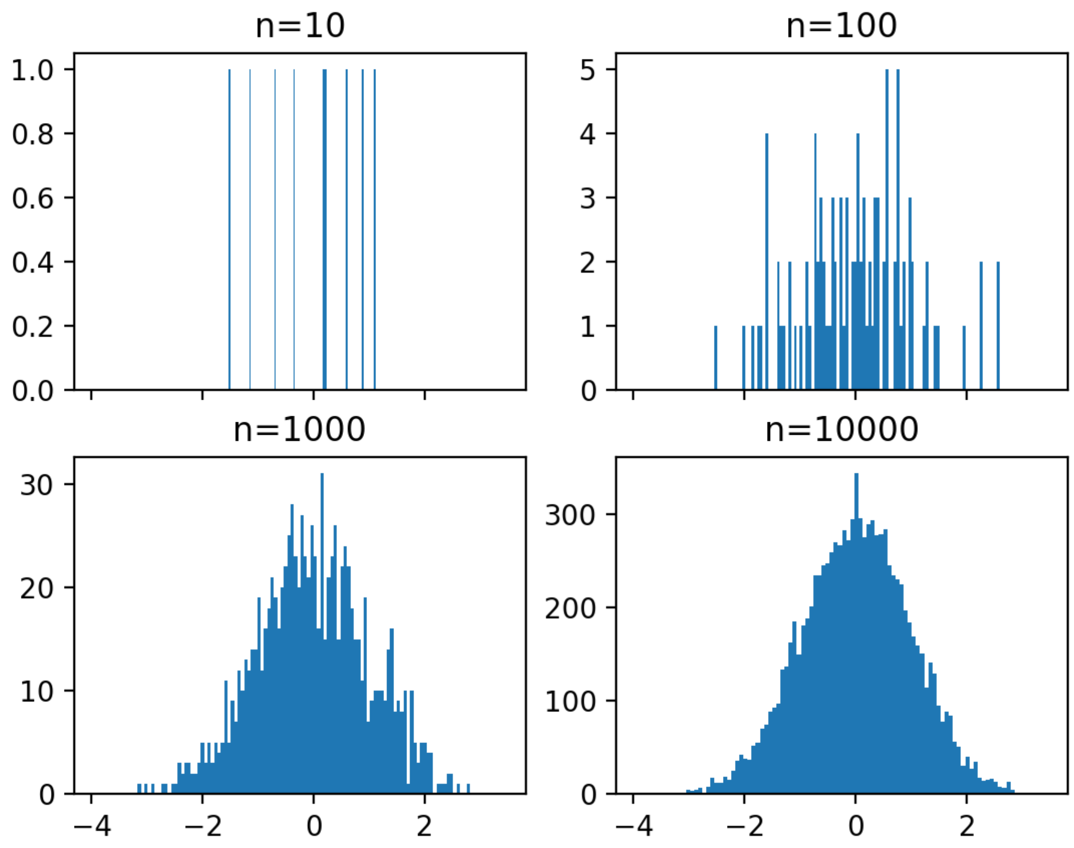
In [10]:
```
# create 2x2 grid of axis subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True)
axs = [ax1,ax2,ax3,ax4]

# draw n = 10, 100, 1000, and 10000 samples from the normal distribution and plot corresponding histograms
for n in range(0,len(axs)):
    sample_size = 10**(n+1)
    sample = np.random.normal(loc=0.0, scale=1.0, size=sample_size)
    axs[n].hist(sample)
    axs[n].set_title('n={}'.format(sample_size))
```

Figure 5


n=10
n=100
n=1000
n=10000
2.0
1.5
1.0
0.5
0.0
20
15
10
5
0
200
150
100
50
0
3000
2000
1000
0
−4
−2
0
2
4

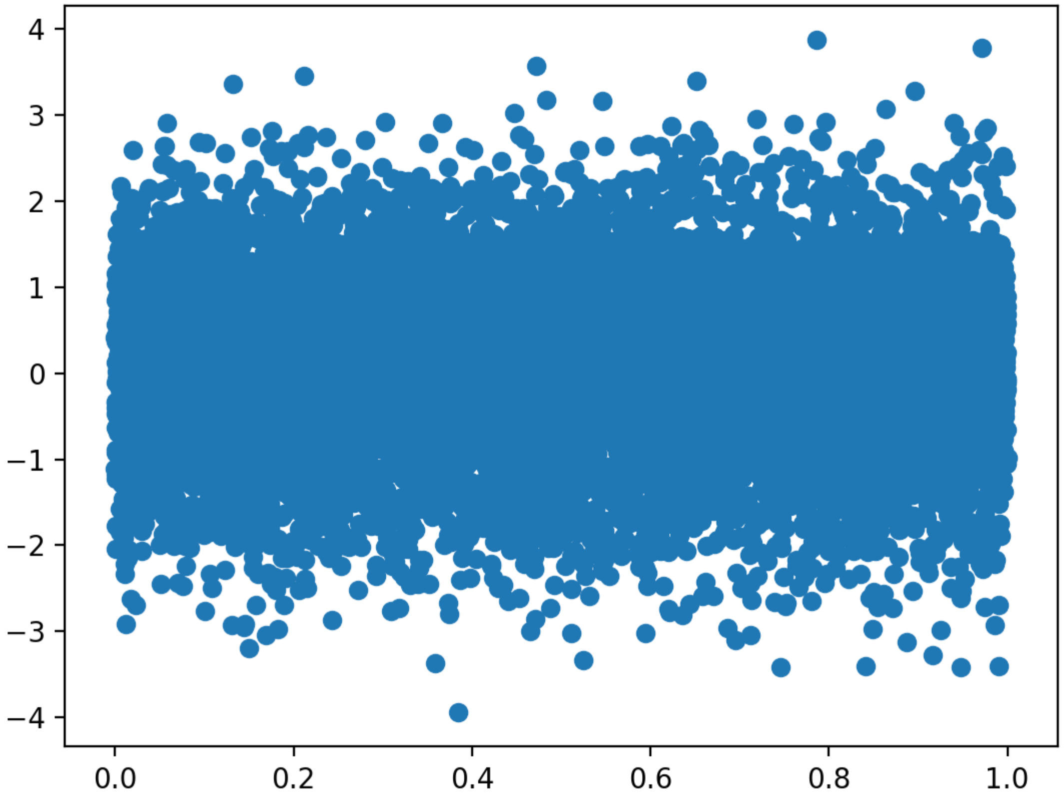
In [11]:
```
# repeat with number of bins set to 100
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex=True)
axs = [ax1,ax2,ax3,ax4]

for n in range(0,len(axs)):
    sample_size = 10**(n+1)
    sample = np.random.normal(loc=0.0, scale=1.0, size=sample_size)
    axs[n].hist(sample, bins=100)
    axs[n].set_title('n={}'.format(sample_size))
```

Figure 6


n=10
n=100
n=1000
n=10000
1.0
0.8
0.6
0.4
0.2
0.0
5
4
3
2
1
0
30
20
10
0
300
200
100
0
−4
−2
0
2
−4
−2
0
2

In [12]:
```
plt.figure()
Y = np.random.normal(loc=0.0, scale=1.0, size=10000)
X = np.random.random(size=10000)
plt.scatter(X,Y)
```

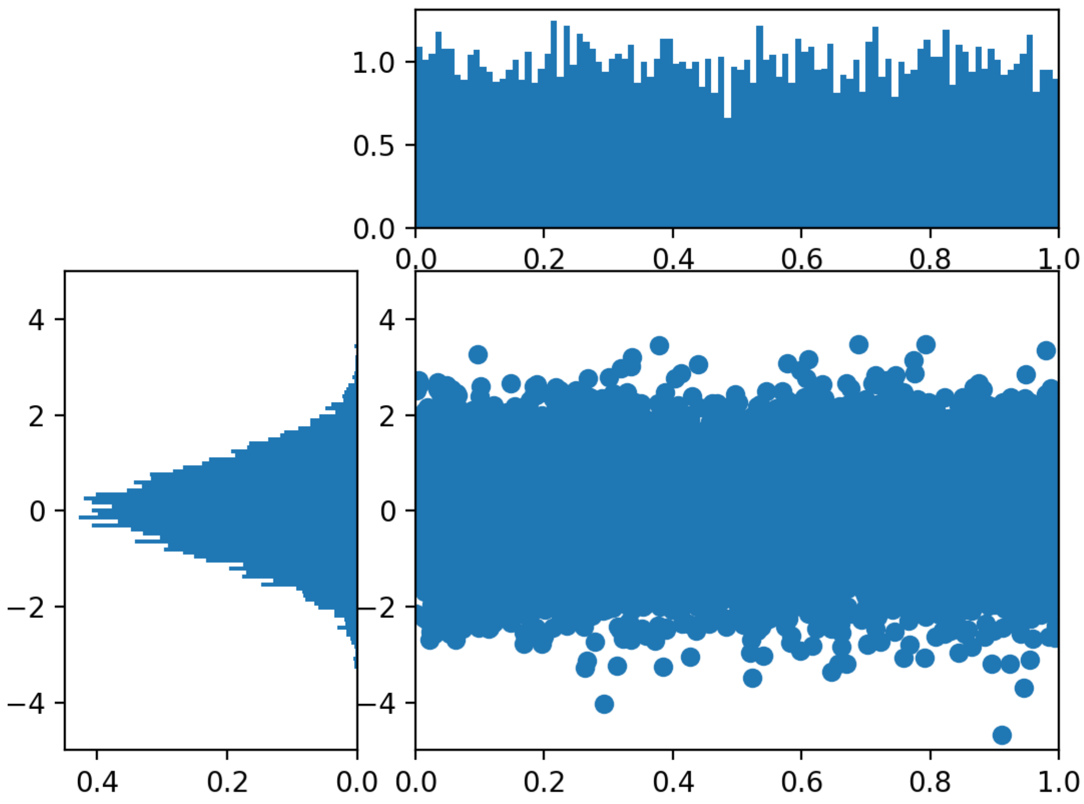
Figure 7



Out[12]: `<matplotlib.collections.PathCollection at 0x7f6340c99e48>`

In [13]:
```
# use gridspec to partition the figure into subplots
import matplotlib.gridspec as gridspec

plt.figure()
gspec = gridspec.GridSpec(3, 3)

top_histogram = plt.subplot(gspec[0, 1:])
side_histogram = plt.subplot(gspec[1:, 0])
lower_right = plt.subplot(gspec[1:, 1:])
```

Figure 8

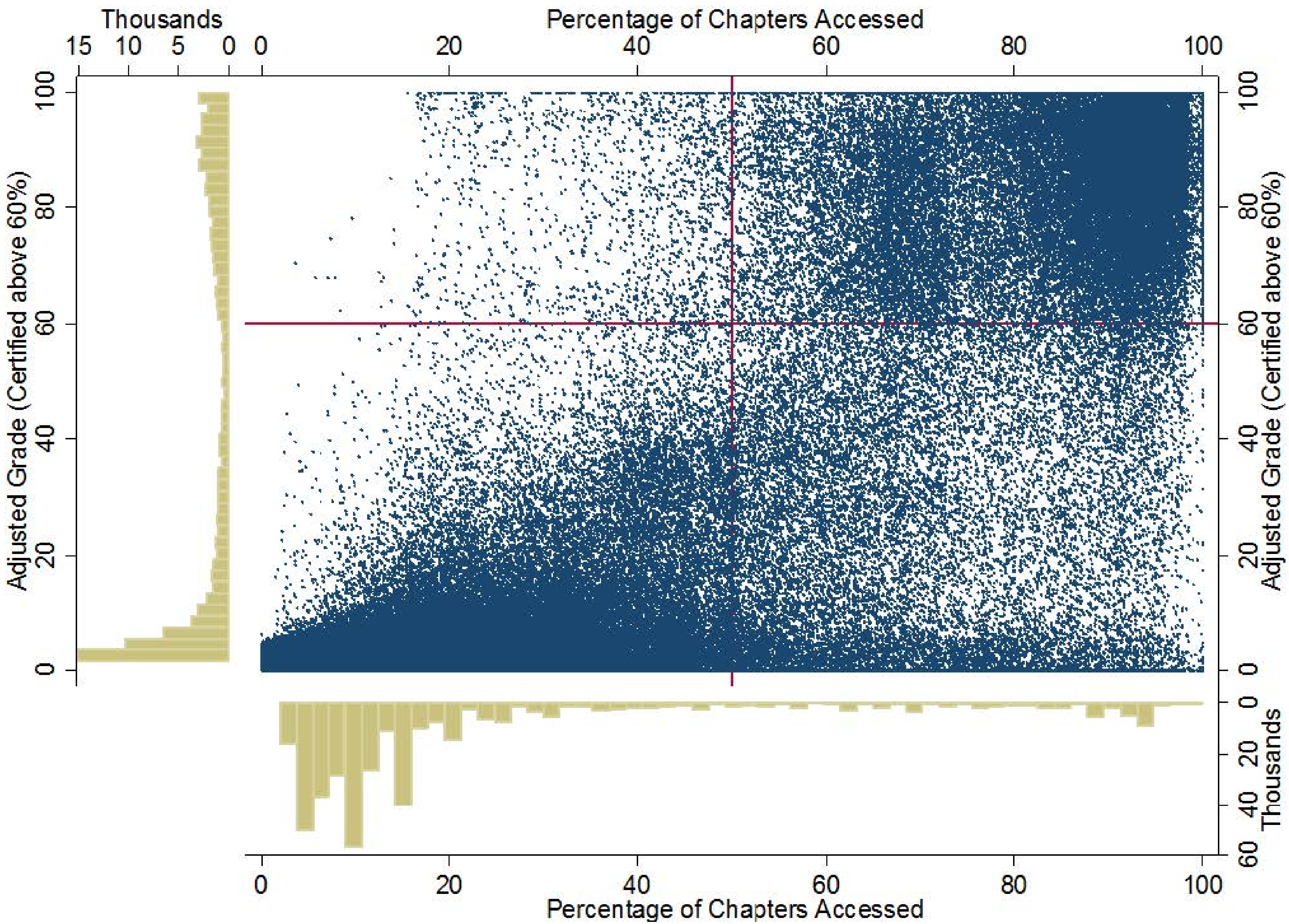


In [14]:
```
Y = np.random.normal(loc=0.0, scale=1.0, size=10000)
X = np.random.random(size=10000)
lower_right.scatter(X, Y)
top_histogram.hist(X, bins=100)
s = side_histogram.hist(Y, bins=100, orientation='horizontal')
```

In [15]:
```
# clear the histograms and plot normed histograms
top_histogram.clear()
top_histogram.hist(X, bins=100, normed=True)
side_histogram.clear()
side_histogram.hist(Y, bins=100, orientation='horizontal', normed=True)
# flip the side histogram's x axis
side_histogram.invert_xaxis()
```

In [16]:
```
# change axes limits
for ax in [top_histogram, lower_right]:
    ax.set_xlim(0, 1)
for ax in [side_histogram, lower_right]:
    ax.set_ylim(-5, 5)
```

In [17]:
```
%%HTML
<img src='http://educationxpress.mit.edu/sites/default/files/journal/WP1-Fig13.jpg' />
```



Box and Whisker Plots

In [18]:
```
import pandas as pd
normal_sample = np.random.normal(loc=0.0, scale=1.0, size=10000)
random_sample = np.random.random(size=10000)
gamma_sample = np.random.gamma(2, size=10000)

df = pd.DataFrame({'normal': normal_sample,
                   'random': random_sample,
                   'gamma': gamma_sample})
```

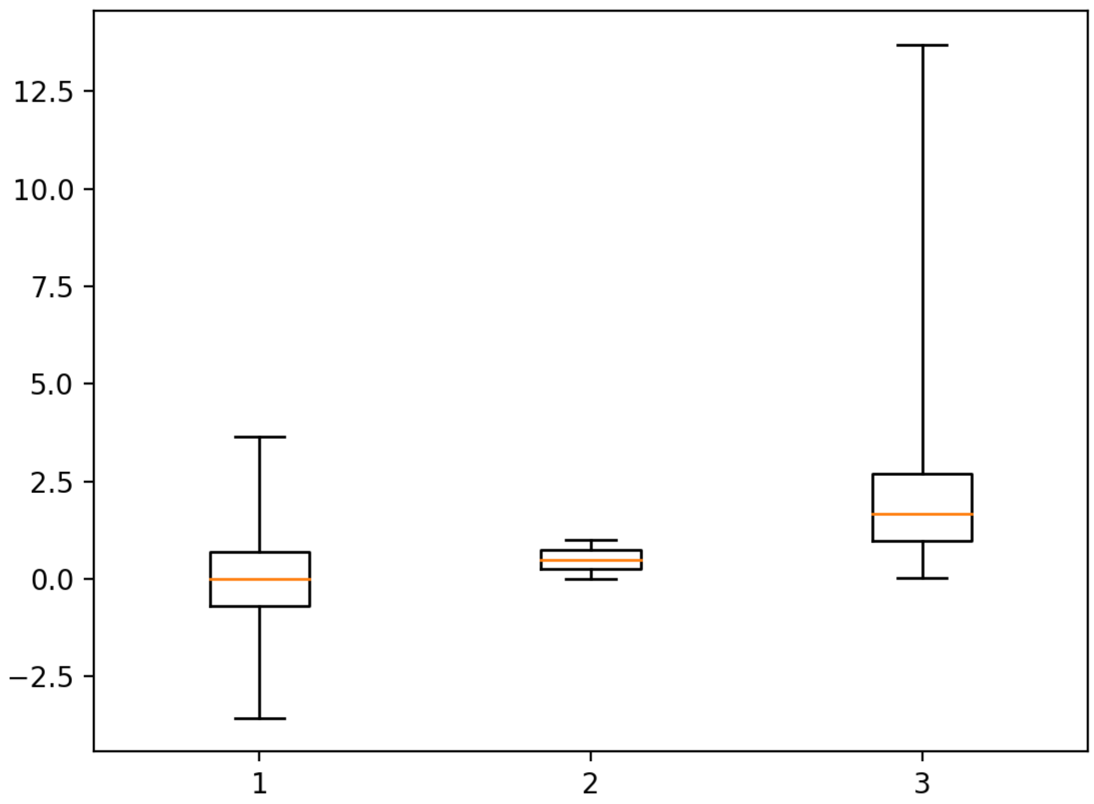
In [19]:
```
df.describe()
```

Out[19]:

	gamma	normal	random
count	10000.000000	10000.000000	10000.000000
mean	2.005316	0.008159	0.500327
std	1.416976	1.011527	0.287147
min	0.016722	-3.564239	0.000033
25%	0.976603	-0.682910	0.258326
50%	1.665339	0.010409	0.497947
75%	2.697683	0.703853	0.748000
max	13.697816	3.649052	0.999954

In [20]:
```
plt.figure()
# create a boxplot of the normal data, assign the output to a variable to supress output
_ = plt.boxplot(df['normal'], whis='range')
```

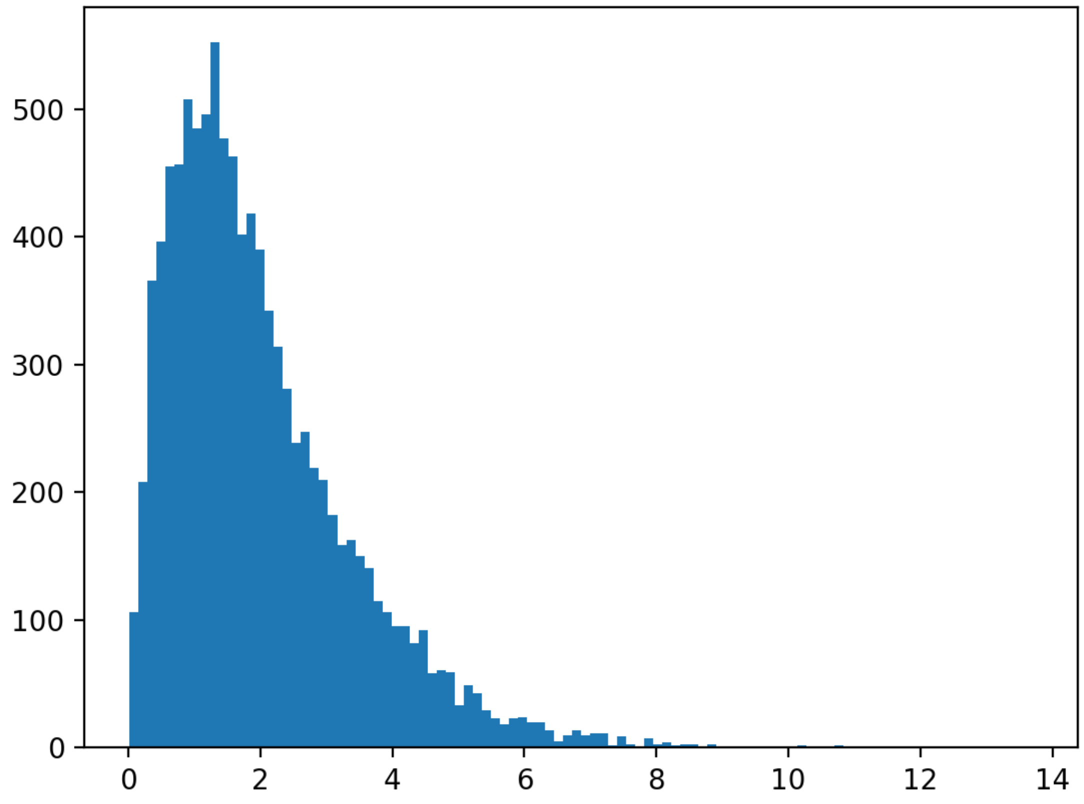
Figure 9



In [21]:
```
# clear the current figure
plt.clf()
# plot boxplots for all three of df's columns
_ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ], whis='range')
```

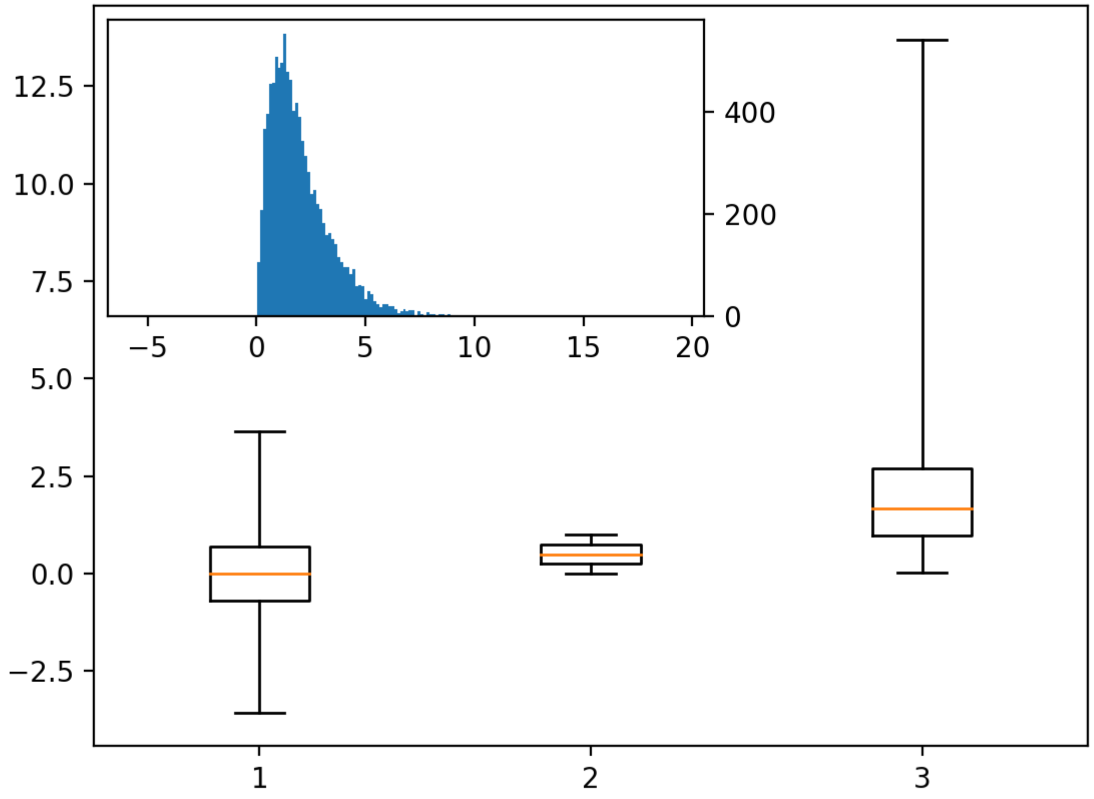
In [22]:
```
plt.figure()
_ = plt.hist(df['gamma'], bins=100)
```

Figure 10



In [23]:
```
import mpl_toolkits.axes_grid1.inset_locator as mpl_il

plt.figure()
plt.boxplot([ df['normal'], df['random'], df['gamma'] ], whis='range')
# overlay axis on top of another
ax2 = mpl_il.inset_axes(plt.gca(), width='60%', height='40%', loc=2)
ax2.hist(df['gamma'], bins=100)
ax2.margins(x=0.5)
```

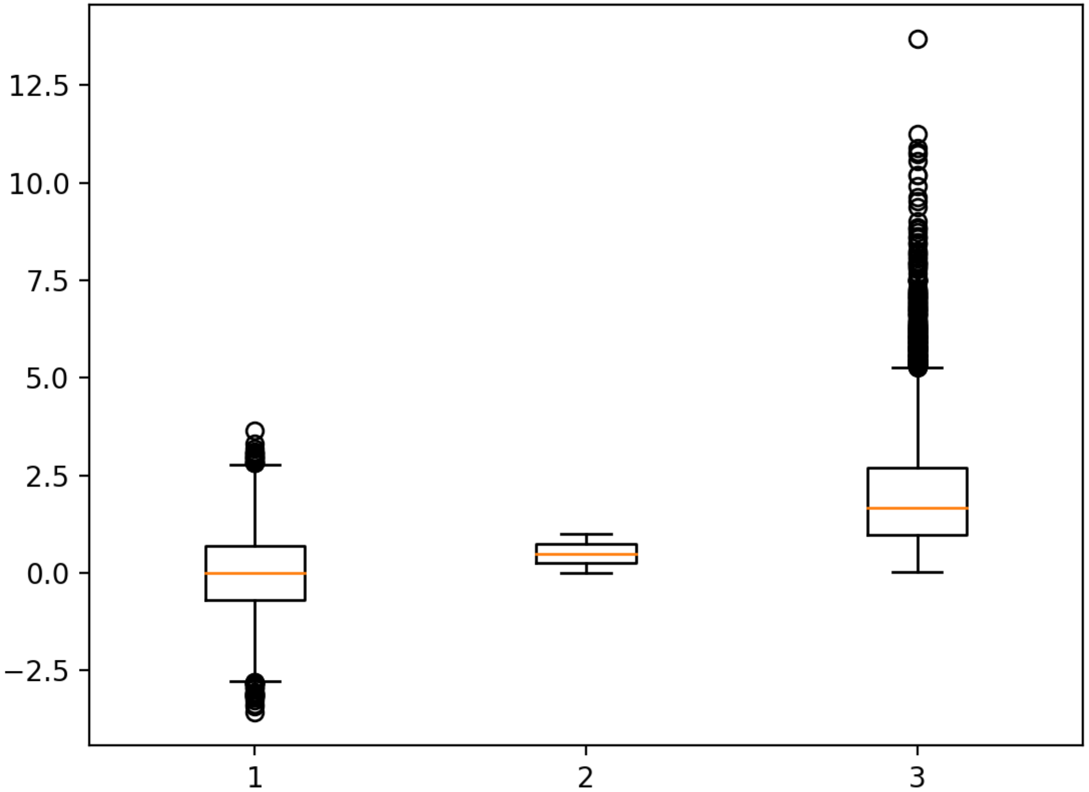
Figure 11



In [24]:
```
# switch the y axis ticks for ax2 to the right side
ax2.yaxis.tick_right()
```

In [25]:
```
# if `whis` argument isn't passed, boxplot defaults to showing 1.5*interquartile (IQR) whiskers with outliers
plt.figure()
_ = plt.boxplot([ df['normal'], df['random'], df['gamma'] ] )
```

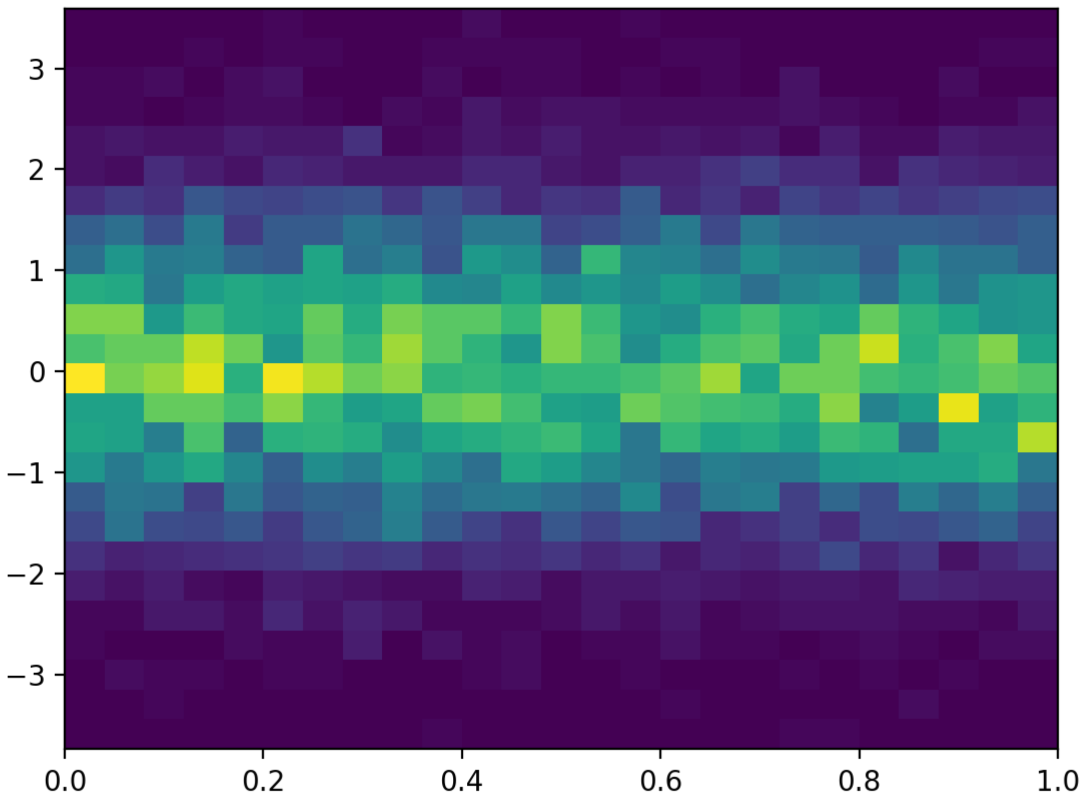
Figure 12



Heatmaps

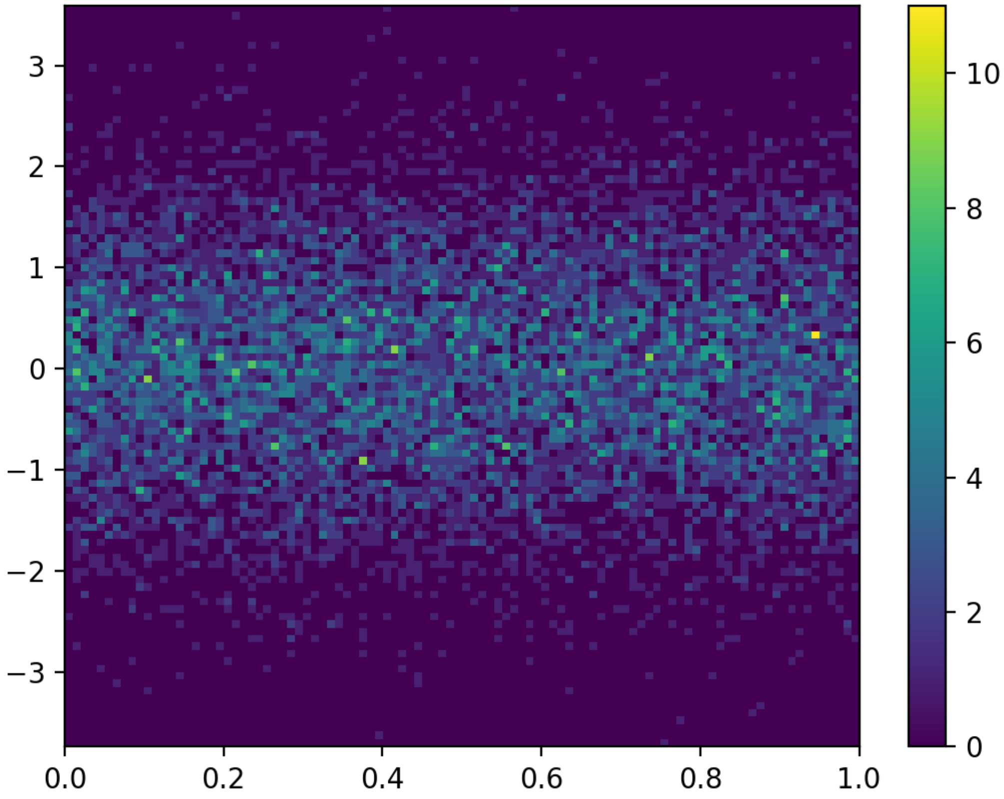
In [26]:
```
plt.figure()

Y = np.random.normal(loc=0.0, scale=1.0, size=10000)
X = np.random.random(size=10000)
_ = plt.hist2d(X, Y, bins=25)
```

Figure 13



In [27]:
```
plt.figure()
_ = plt.hist2d(X, Y, bins=100)
```

Figure 14



In [28]:
```
# add a colorbar legend
plt.colorbar()
```

Out[28]: <matplotlib.colorbar.Colorbar at 0x7f63381f6c88>

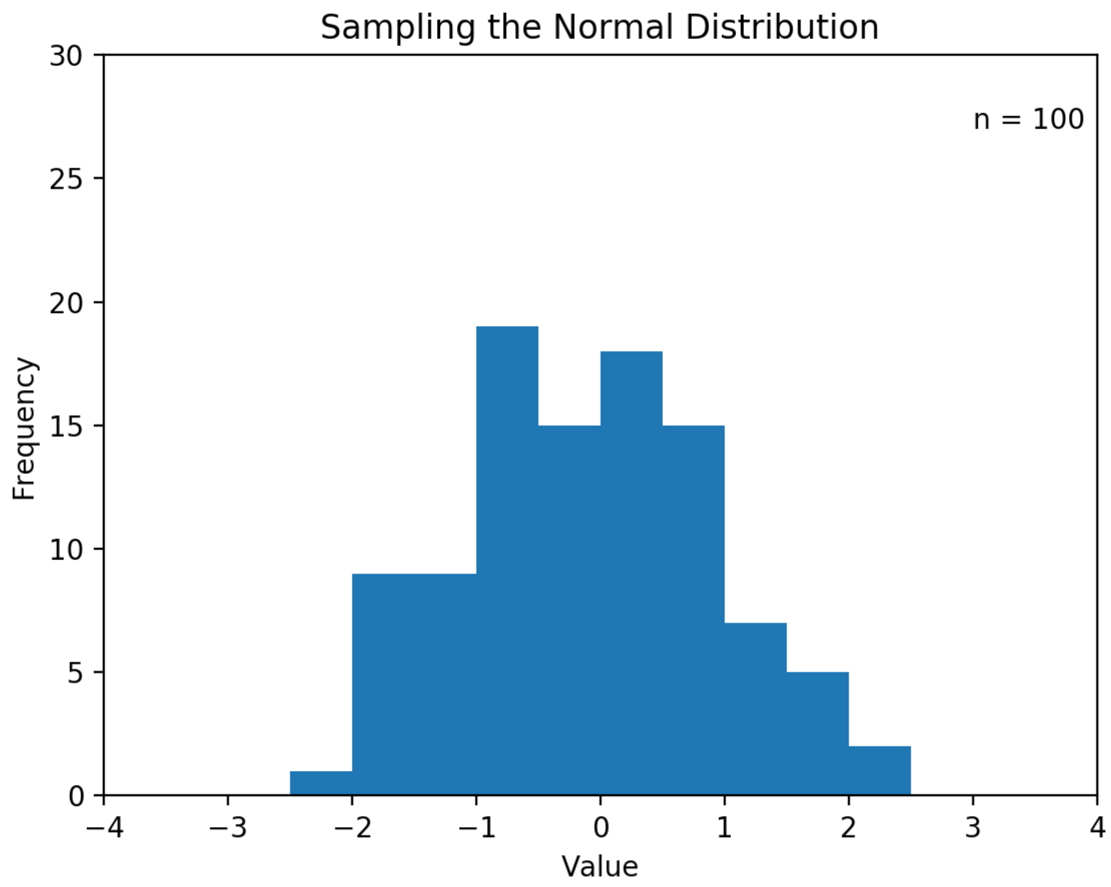
Animations

In [29]:
```
import matplotlib.animation as animation

n = 100
x = np.random.randn(n)
```

In [30]:
```
# create the function that will do the plotting, where curr is the current frame
def update(curr):
    # check if animation is at the last frame, and if so, stop the animation a
    if curr == n:
        a.event_source.stop()
    plt.cla()
    bins = np.arange(-4, 4, 0.5)
    plt.hist(x[:curr], bins=bins)
    plt.axis([-4,4,0,30])
    plt.gca().set_title('Sampling the Normal Distribution')
    plt.gca().set_ylabel('Frequency')
    plt.gca().set_xlabel('Value')
    plt.annotate('n = {}'.format(curr), [3,27])
```

In [36]:
```
fig = plt.figure()
a = animation.FuncAnimation(fig, update, interval=100)
```

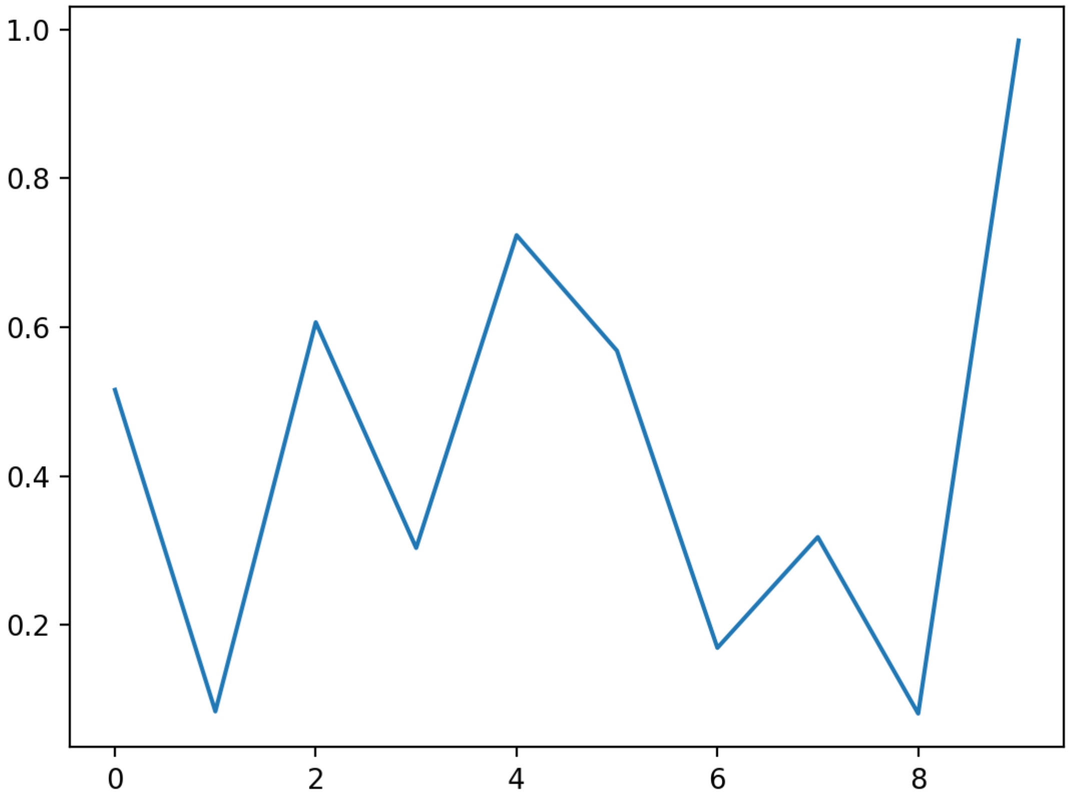
Figure 18



Interactivity

In [32]:
```
plt.figure()
data = np.random.rand(10)
plt.plot(data)

def onclick(event):
    plt.cla()
    plt.plot(data)
    plt.gca().set_title('Event at pixels {},{} \nand data {},{}'.format(event.x, event.y, event.xdata, event.yda

# tell mpl_connect we want to pass a 'button_press_event' into onclick when the event is detected
plt.gcf().canvas.mpl_connect('button_press_event', onclick)
```

Figure 16



Out[32]: 7

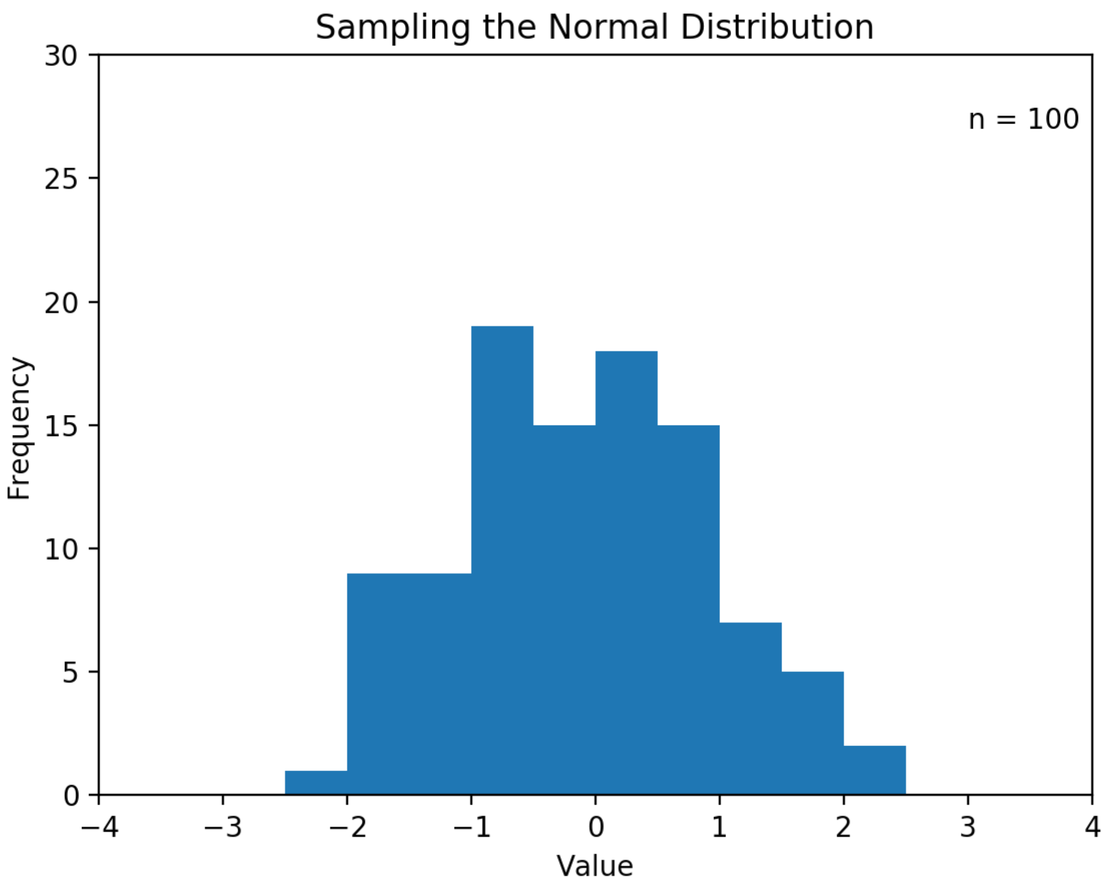
In [33]:
```
from random import shuffle
origins = ['China', 'Brazil', 'India', 'USA', 'Canada', 'UK', 'Germany', 'Iraq', 'Chile', 'Mexico']

shuffle(origins)

df = pd.DataFrame({'height': np.random.rand(10),
                   'weight': np.random.rand(10),
                   'origin': origins})
df
```

Out[33]:

	height	origin	weight
0	0.604175	UK	0.517334
1	0.520595	Brazil	0.834557
2	0.939892	Canada	0.315674
3	0.380012	Mexico	0.003923
4	0.781178	China	0.777982
5	0.305815	USA	0.832483
6	0.599650	India	0.253113
7	0.199662	Chile	0.130700
8	0.474623	Germany	0.345397
9	0.685690	Iraq	0.923716

In [34]:
```
plt.figure()
# picker=5 means the mouse doesn't have to click directly on an event, but can be up to 5 pixels away
plt.scatter(df['height'], df['weight'], picker=5)
plt.gca().set_ylabel('Weight')
plt.gca().set_xlabel('Height')
```

Figure 17



Out[34]: <matplotlib.text.Text at 0x7f6338194240>

In [35]:
```
def onpick(event):
    origin = df.iloc[event.ind[0]]['origin']
    plt.gca().set_title('Selected item came from {}'.format(origin))

# tell mpl_connect we want to pass a 'pick_event' into onpick when the event is detected
plt.gcf().canvas.mpl_connect('pick_event', onpick)
```

Out[35]: 7